Actividad grupal. Desarrollo de un problema de planificación

Resumen

Introducción

Definición de planificación

Según la RAE se le define a la planificación como aquel plan metódicamente organizado y de gran amplitud para la obtención de un objetivo determinado. (Asale, s. f.) Por otro lado, la planificación en IA es una de sus áreas encargadas de, mediante programación automática, diseñar planes de acción que serán ejecutadas por un agente inteligente para la obtención de una meta o resultado deseado. (Vazquez, 2012).

Se busca dotar a un agente inteligente con:

* La representación del objetivo que se va a alcanzar.
* La representación de las acciones que puede realizar.
* La representación del entorno.
* La capacidad de generar un plan para alcanzar el objetivo.

Problema de planificación

Un problema de planificación consiste en un estado inicial el cual mediante la aplicación de una secuencia de acciones llevan a un estado final u objetivo. Se considera a la planificación como un problema de búsqueda en un espacio de estados. A continuación, se muestra un grafo que representa un problema de planeación (Ilustración 1), donde delta sub cero es el estado inicial, delta sub-n es el estado final y delta representa la acción que se ejecuta para pasar de un estado (nodo) a otro (Agentes planificadores., s. f.).

A diagram of a network

Description automatically generated

Ilustración Representación gráfica de un problema de planificación

Tipos de planes

Existen dos tipos de planes: los planes parcialmente ordenados y los totalmente ordenados. Los planes parcialmente ordenados consisten en un conjunto de acciones ordenadas en las cuales se tienen restricciones en cuanto a su secuencia de ejecución, donde pueden haber más de dos ramificaciones que son ejecutadas dependiendo de la opción que se elija. Los planes totalmente ordenados consisten en acciones las cuales se van ejecutando una tras otra siguiendo una secuencia lineal (Agentes planificadores., s. f.).

Algoritmo de planificación

Un algoritmo de planificación enlista todos los estados intermedios que llevan del estado inicial al final, reduciendo su complejidad descomponiendo el problema en subobjetivos (Agentes planificadores., s. f.).

Planificación clásica

Se emplea la planificación clásica donde los entornos son:

* Completamente observables para el planificador, este percibe todo el estado, sus propiedades y el efecto de las acciones.
* Son deterministas porque se puede predecir y predefinir los efectos de las acciones que se ejecutan.
* Son finitos sus estados, acciones y efectos.
* Son estáticos, ya que cambian solo cuando el planificador actúa sobre él.
* Son discretos, es decir, son contables tanto sus acciones, objetos y efectos.

PDDL

Para programar estos planificadores se emplea el lenguaje PDDL (Planning Domain Definition Language), estándar usado para la representación de tareas de planificación clásica usado desde 1998 por el ICAPS (International Conference on Automated Planning and Scheduling). Se inspiró en UCPOP, el planificador desarrollado por la Universidad de Washington. (Berzal).

International Planning Competition

Esta competición se ha llevado a cabo desde 1998 en varias partes del mundo por el ICAPS (Competitions – ICAPS, s. f.), presenta varias pistas o tracks casi cada año, las cuales son:

* Deterministic.
* Probabilistic.
* Uncertainty.
* Learning.
* Probabilistic Continuous.
* Probabilistic Discrete.
* Temporal.
* Classical.
* Numeric.
* HTN

Estas pistas cuentan con sub categóricas en las que se registran varios equipos los cuales deben darle solución a una de varias tareas planteadas usando planificadores, estos planificadores de cumplir con una serie de condiciones propias de dicha subcategoría como el tiempo de ejecución, el límite de memoria usado, el CPU usado, el número de tareas resuelto entre otros (Competitions – ICAPS, s. f.). Por ejemplo, algunas de las tareas que se han usado en esta competición son:

* El juego de cartas Agrícola.
* Ciberseguridad.
* Procesamiento y envío de datos en una red.
* El puzzle Nurikabe.
* El juego Snake.
* El juego Spider.
* Laberinto.
* Cubo Rubik.
* Síntesis de diseño de círculos cuánticos.

Desarrollo

Para realizar esta actividad, primero se hizo la instalación de Singularity en dos computadoras, una computadora con sistema operativo Linux y otra en una máquina virtual con Ubuntu v22.04.3 (Ilustración 2).

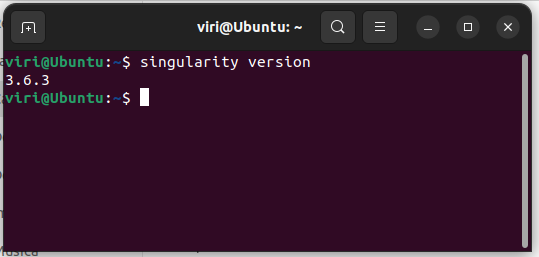


Ilustración Singularity versión

* Pruebas de ejecución del planner ganador y los archivos domain y problem de la Tarea Snake

Una vez instalado Singularity se tomó el repositorio del equipo 23 Delfi1 participante y ganador en la IPC 2018 (International Planning Competition 2018) para la generación del archivo Singularity (IPC 2018, 2018). Se ejecutó en una terminal la línea de comando *wget* [*https://bitbucket.org/ipc2018-classical/team23/raw/ipc-2018-seq-opt/Singularity*](https://bitbucket.org/ipc2018-classical/team23/raw/ipc-2018-seq-opt/Singularity) (Ilustración 3).

A screenshot of a computer

Description automatically generated

Ilustración Generación del archivo Singularity a partir del repositorio

del equipo 23 Delfi1

Luego en la misma terminal, con el comando *sudo**singularity build planner.img Singularity* se creó la imagen del planificador del repositorio del equipo 23 Delfi1 (Ilustración 4).

A screenshot of a computer

Description automatically generated

Ilustración Se crea el archivo planner.img

La imagen del planificador se creó en la misma carpeta que el archivo Singularity (Ilustración 5), esto es importante ya que ambos archivos deben de estar juntos, de lo contrario al ejecutar los siguientes comandos se va a producir un error.

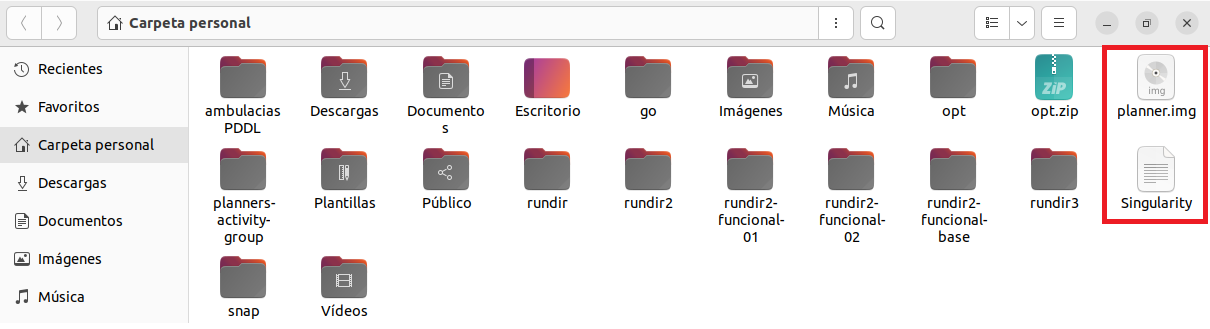


Ilustración Ubicación de los archivos Planner.img y Singularity

Después se toman el archivo domain y uno de los archivos problema de la tarea Snake, se agregan a la carpeta rundir y se ejecutan los siguientes comandos (Ilustración 6):

* + RUNDIR="$(pwd)/rundir": se toma la ubicación de la carpeta que contiene los archivos domain y problem, y se debe de especificar el nombre de dicha carpeta, en este caso la carpeta se llama “rundir”.
  + DOMAIN="$RUNDIR/domain.pddl": se especifica el nombre del archivo del dominio, en este caso se llama domain.
  + PROBLEM="$RUNDIR/problem.pddl": se especifica el nombre del archivo del problema, en este caso se llama problem.
  + PLANFILE="$RUNDIR/sas\_plan": Se especifica la ubicación en donde se va a generar el archivo que contendrá el plan generado, en este caso el archivo recibirá el nombre sas\_plan y se guardará en la misma carpeta que aloja los archivos del dominio y del problema.
  + ulimit -t 1800: se define el tiempo límite.
  + ulimit -v 8388608: se define el volumen límite.
  + singularity run -C -H $RUNDIR planner.img $DOMAIN $PROBLEM $PLANFILE $COSTBOUND: este comando ejecuta el archivo domain y problem usando el archivo Singularity y planner.

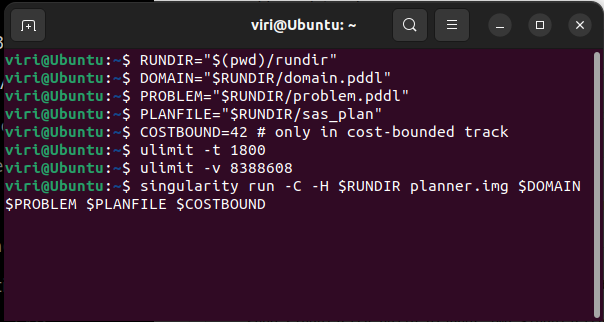


Ilustración Comandos para la ejecución del domain y problem de snake en una máquina virtual

A continuación, se muestra la ejecución de los archivos domain y problem de la tarea Snake (Ilustración 7):

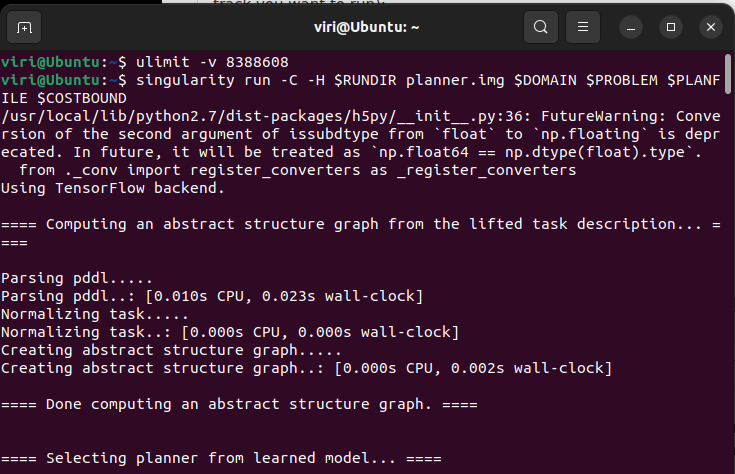


Ilustración 7 Ejecución del domain y problem de snake

A screenshot of a computer program

Description automatically generated

Ilustración 8 Finalización de ejecución de domain y problem de snake

Una vez terminada la ejecución se genera el plan de la tarea y es guardado en el archivo sas\_plan, el cual es almacenado en la carpeta “rundir”, a continuación, se muestra el plan generado para la tarea Snake (Ilustración 9).

A screenshot of a computer program

Description automatically generated

Ilustración Se muestra el plan creado tras la ejecución del script de Singularity

* Descripción de escenario original de la actividad

Una vez realizada la ejecución del planificador se procede a trabajar con el siguiente problema planteado:

Se tiene un robot tipo *rover* que previamente realizó la excavación de dos rocas localizadas en la localidad 1 y localidad 2. Debido al mal tiempo, no fue posible trasladar las rocas para su análisis. Es por ello por lo que se solicita generar el plan que debe seguir el robot para llevar los minerales al laboratorio de análisis.

Debido al terreno, hay restricciones en la trayectoria: de la localidad 3 a la 1, el camino está libre y existe bidireccionalidad; de la localidad 3 a la 2, el camino solo es de una dirección; de la localidad 2 a la 4 es solo una dirección; de la localidad 3 a 4 y de 4 a 5, el camino es bidireccional. El robot solo puede cargar un mineral a la vez.

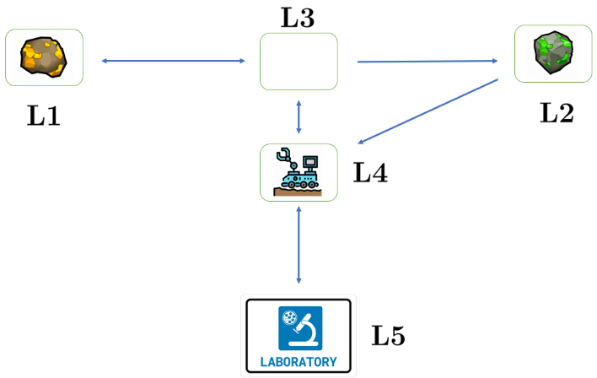


Ilustración Esquema del problema

Para solucionar esta tarea se creó en un archivo .pddl un dominio llamado robot-rover, se definieron los objetos robot, location y state, para el objeto state se definieron 2 tipos: mineral y laboratory (Ilustración 11).

A computer screen shot of text

Description automatically generated

Ilustración Definición del dominio y objetos

Luego se definieron los predicados (Ilustración 12):

* + (**rover ?r - robot**) : el predicado rover recibe la variable **r** que es de tipo robot e indica que **r** es un robot.
  + (**location ?field - location**) : el predicado location recibe la variable **field** y es de tipo location e indica que **field** es una ubicación.
  + (**mineral ?m - mineral**) : el predicado mineral recibe la variable **m** y es de tipo mineral e indica que **m** es un mineral.
  + (**lab ?lab - laboratory**) : el predicado lab recibe la variable **lab** y es de tipo laboratory e indica que **lab** es un laboratorio.
  + (**there\_a\_mineral ?rock – mineral ?field - location**): el predicado there\_a\_mineral recibe las variables **rock** de tipo mineral y **field** de tipo location e indica que un mineral dado está en la ubicación dada.
  + (**there\_a\_lab ?lab – laboratory ?field - location**): el predicado there\_a\_lab recibe las variables **lab** de tipo laboratory y **field** de tipo location e indica que un laboratorio dado se encuentra está en la ubicación dada.
  + (**rover\_is\_charged ?rock – mineral ?r - robot**) este predicado recibe las variables **rock** de tipo mineral y **r** de tipo robot e indica que un mineral se encuentra en el robot.
  + (**at ?field - location**): recibe la variable **field** de tipo location e indica la posición actual donde se encuentra el robot.
  + (**rover\_is\_discharge ?r - robot**) este predicado recibe la variable **r** de tipo robot e indica que rover está libre, no trae ningún mineral.
  + (**move ?from ?to - location**) este predicado recibe las variables **from** y **to** de tipo location e indican el desplazamiento del robot de un punto a A un punto B.

A computer code with black text

Description automatically generated

Ilustración Declaración de los predicados

A continuación, se definen las acciones que el planificador llevara a cabo para solucionar esta tarea. Se crea la acción move\_to\_other\_location la cual recibe como parámetros la variable **r** de tipo robot y las variables **from** y **to** de tipo location. Como precondiciones se declara que hay un robot, una ubicación **from** (desde) y una ubicación **to** (hasta), el robot se encuentra en una ubicación especifica y se desplaza desde esa ubicación a otra. Por tanto, los efectos serían que el robot se encuentra ahora en la ubicación hasta (destino) y ya no está en la ubicación desde (inicio) (Ilustración 13).

A computer screen shot of a program code

Description automatically generated

Ilustración Definición de la acción move\_to\_other\_location

Se declara la acción mineral\_found que recibe los parámetros **r** de tipo robot, **from** de tipo location y **m** de tipo mineral. Como precondiciones se define que hay un mineral en una ubicación dada donde se encuentra el robot y el robot está descargado. Esto da como efecto que el robot recoge este mineral y ahora lo estará cargando, el mineral ya no se encuentra en la ubicación y el robot ya no está libre.

A screenshot of a computer code

Description automatically generated

Ilustración Definición de la acción mineral\_found

Por último, se define la acción mineral\_discharge la cual recibe los parámetros **r** de tipo robot, **from** de tipo location, **m** de tipo mineral y **la** de tipo laboratory. Como precondiciones se define que en una ubicación dada donde hay un laboratorio se encuentra el robot el cual trae el mineral. Y como efecto se tiene como respuesta que hay un mineral en esa ubicación y el robot está libre.

A screenshot of a computer program

Description automatically generated

Ilustración Se define la acción mineral\_discharge

Se declara el archivo .pddl problem llamado rover-the-robot-initial-prob, se especifica el nombre del dominio con el que se va a ejecutar en conjunto y se declaran los objetos que formaran parte del entorno (Ilustración 16).

A screenshot of a computer

Description automatically generated

Ilustración Definición del archivo problem

Se declara el estado inicial de los objetos en el entorno, es decir, se inicializan los minerales que se encuentran en el mapa, el robot y su estado inicial que es libre, las ubicaciones que conforman el mapa, el laboratorio y las ubicaciones iniciales en donde se encuentran los minerales, el laboratorio y el robot. También se definen los movimientos que son válidos por donde puede ir el robot (Ilustración 17). Todo esto se define según lo especificado en la descripción del problema planteado (Ilustración 10).

A screenshot of a computer program

Description automatically generated

Ilustración Definición del estado inicial de los objetos

Por último, se define el estado final u objetivo de los objetos el cual debe ser que ambos minerales deben de estar ubicados en el laboratorio, es decir la ubicación L5 (Ilustración 18).

A black text on a white background

Description automatically generated

Ilustración Definición del estado final u objetivo

Se muestra la ejecución de ambos archivos desarrollados, se usa la imagen del planner y el script generado a partir del planner presentado por el equipo 23 Delfi1 (Ilustración 19 y 20).

A screenshot of a computer program

Description automatically generated

Ilustración Fin de la ejecución de los archivos domain y problem desarrollados

A computer screen shot of a computer program

Description automatically generated

Ilustración Inicio de la ejecución del archivo domain y problem

A diagram of a laboratory

Description automatically generatedA screenshot of a computer

Description automatically generatedSe muestra el plan generado a partir de los archivos domain y problem desarrollados (Ilustración 21). Como se puede observar el plan coincide con el esquema del problema planteado (Ilustración 22).

Ilustración Plan generado

Ilustración Esquema planteado del problema

* Descripción de escenario 1 propuesto

A continuación, se propone un escenario alterno en donde se toma como base el archivo domain mostrado y explicado anteriormente. En este nuevo archivo .pddl se define el problema rover-the-robot-initial-prob-alt se plantea la siguiente situación (Ilustración 23, 24 y 25):

* + Se tienen 3 minerales: yellowrock, greenrock y redrock.
  + Hay 3 laboratorios: lab1, lab2 y lab3.
  + Se agregan las ubicaciones L6, L7, L8 y L9.

A screenshot of a computer

Description automatically generated

Ilustración Se define el problema rover-the-robot-initial-prob-alt

* + Las ubicaciones agregadas tienen las siguientes restricciones:
    - Se puede ir de L3 a L6.
    - Se puede ir de L1 a L6 y de L6 a L1.
    - Se puede ir de L8 a L4 y de L4 a L8.
    - Se puede ir de L8 a L1.
    - Se puede ir de L6 a L9 y de L9 a L6.
    - Se puede ir de L6 a L7.
    - Se puede ir de L7 a L2.

A screenshot of a computer program

Description automatically generated

Ilustración Estado inicial de los objetos

* + Al final el robot debe de ir a la ubicación L7.
  + Cada mineral debe ser transportado a su laboratorio correspondiente:
    - A black text on a white background

      Description automatically generatedGreenrock al laboratorio lab2.
    - Redrock al laboratorio lab3.
    - Yellowrock al laboratorio lab1.

Ilustración Estado final de los objetos

Se muestra la ejecución del archivo domain y el nuevo archivo problem1, se usa la imagen del planner y el script generado a partir del planner presentado por el equipo 23 Delfi1 (Ilustración 26 y 27).

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

Ilustración Fin de la ejecución del archivo domain y problem1

Ilustración Inicio de la ejecución de los archivos domain y problem1

Se muestra el plan generado a partir de los archivos domain y problem1 (Ilustración 28). Como se puede observar el plan coincide con el nuevo esquema del problema planteado (Ilustración 29).

A screenshot of a video game

Description automatically generatedA screenshot of a computer

Description automatically generated

Ilustración Esquema generado para el nuevo problema planteado

Ilustración Plan generado usando el archivo problem1

* Descripción de escenario 2 propuesto

Dificultades encontradas

Algunas de las dificultades encontradas durante la realización de esta actividad fueron:

* Entender la lógica de un planner, ya que la documentación que hay es escasa y solo explica lo básico, la única documentación que presenta un poco más de información es el libro An Introduction to the Planning Domain Definition Language, pero no hay más alternativas si se quiere ahondar más sobre el tema.
* Tampoco se cuentan con ejemplos completos que puedan ser replicados para ver el plan generado tras ser ejecutado el planner y entender más sobre su funcionamiento.
* Hubo dificultades al instalar Singularity ya que algunos comandos usados para su instalación no funcionaban y generaban errores, por lo cual se tenía que buscar como instalar paqueterías adicionales que al final no resolvían el error marcado. Al final se encontró un post en GitHub en el que se explica una alternativa para instalarlo (Apptainer, s. f.).
* Se tuvieron problemas al ejecutar los planners ya que se presentaban errores del código por paréntesis de más, la falta de alguna variable en un predicado que debe de llevar dos y solo se le estaba asignando una, error por el nombre incorrecto de alguna variable.
* Se tuvieron que modificar en varias ocasiones los archivos domain y problem porque no se generaba el plan o se generaba distinto si se comparaba con el plan generado tras ejecutar los archivos desde la terminal (usando la imagen del planner de algún equipo) con el plan generado a través del editor de archivos pddl.

Conclusiones

El desarrollo de planificadores es interesante es una lástima que no sea un área tan explotada como las demás áreas de la IA, ya que de momento solo se implementa en empresas grandes, agencias y en la conferencia de la ICAPS, esto es entendible ya que para su ejecución se necesita de un gran gasto computacional, además de que no se cuenta con mucha documentación para estudiar más sobre su desarrollo lo que supone un reto mayor para aquellos interesados en este tema. Se espera que en un futuro cercano se pueda implementar en otras áreas dado que el coste computacional sería menor por el uso de computadoras con mayores recursos.

Tras realizar esta actividad se aprendió sobre la programación de planificadores, su estructura, codificación y ejecución. Se aprendió sobre los tipos de planificadores, la planificación clásica, se aprendieron definiciones importantes como que es un plan, un planificador, en qué consiste un problema de planificación y un algoritmo de planificación y sobre qué es el lenguaje PDDL.

Referencias Bibliográficas

* Asale, R.-. (s. f.). Planificación | Diccionario de la Lengua Española. «Diccionario de la lengua española» - Edición del Tricentenario. <https://dle.rae.es/planificaci%C3%B3n>
* Vazquez, J. (2012). Obtenido de <https://www.cs.upc.edu/~jvazquez/teaching/iag/transpas/4-PL1-IntroPlan>
* Berzal, F. (s.f.). *DECSAI.* Obtenido de Departamento de Ciencias de la Computación e IA: <https://elvex.ugr.es/decsai/iaio/slides/P9%20PDDL.pdf>
* IPC 2018. (2018). INTERNATIONAL PLANNING COMPETITION 2018. Recuperado 22 de agosto de 2023, de <https://ipc2018-classical.bitbucket.io/>
* Agentes planificadores. (s. f.). [Diapositivas]. Universidad Politécnica de Cataluña. <https://www.cs.upc.edu/~jvazquez/teaching/iag/transpas/4-PL1-IntroPlanificaci%c3%b3n.pdf>
* Competitions – ICAPS. (s. f.). <https://www.icaps-conference.org/competitions/>
* Patrik Haslum; Nir Lipovetzky; Daniele Magazzeni; Christian Muise, *An Introduction to the Planning Domain Definition Language*, Morgan & Claypool, 2019.
* Apptainer. (s. f.). Singularity 3.5.3 mConfig Not recognizing Installed Go · Issue #5099 · Apptainer/Singularity. GitHub. <https://github.com/apptainer/singularity/issues/5099>